EXHIBIT 36

Exhibit 4

Arista eAPI 101

eos.arista.com/arista-eapi-101/
Addison
Chi

In this article, you will get some quick ideas to use Arista eAPI to configure the switch via JSON-RPC remotely. Here is one Youtube video on eAPI if you prefer to watch something live:

- EOS Bits & Bytes – Episode 3 “Command API”

The Arista Command eAPI is a simple and complete API that allows you to configure and monitor your Arista switches. Once the API is enabled, the switch accepts HTTP(S) requests containing a list of industry standard CLI commands, and responds with machine-readable output and errors serialized in JSON (served over HTTP or HTTPS). eAPI was first introduced in EOS version 4.12 and you can download the full eAPI guide in the EOS download folder. Step 1: Configuring the Command API Agent in the switch: Although disabled by default, it is very simple to get the Command API server running on your switch. To modify its configuration, from configure mode, enter “management api http-commands” mode. In this submode, you can turn on or off the server by typing “[no] shutdown”, switch between accepting HTTP or HTTPS traffic via “[no] protocol http[s]”, and adjust the ports the server should listen on using “protocol http[s] port ”.

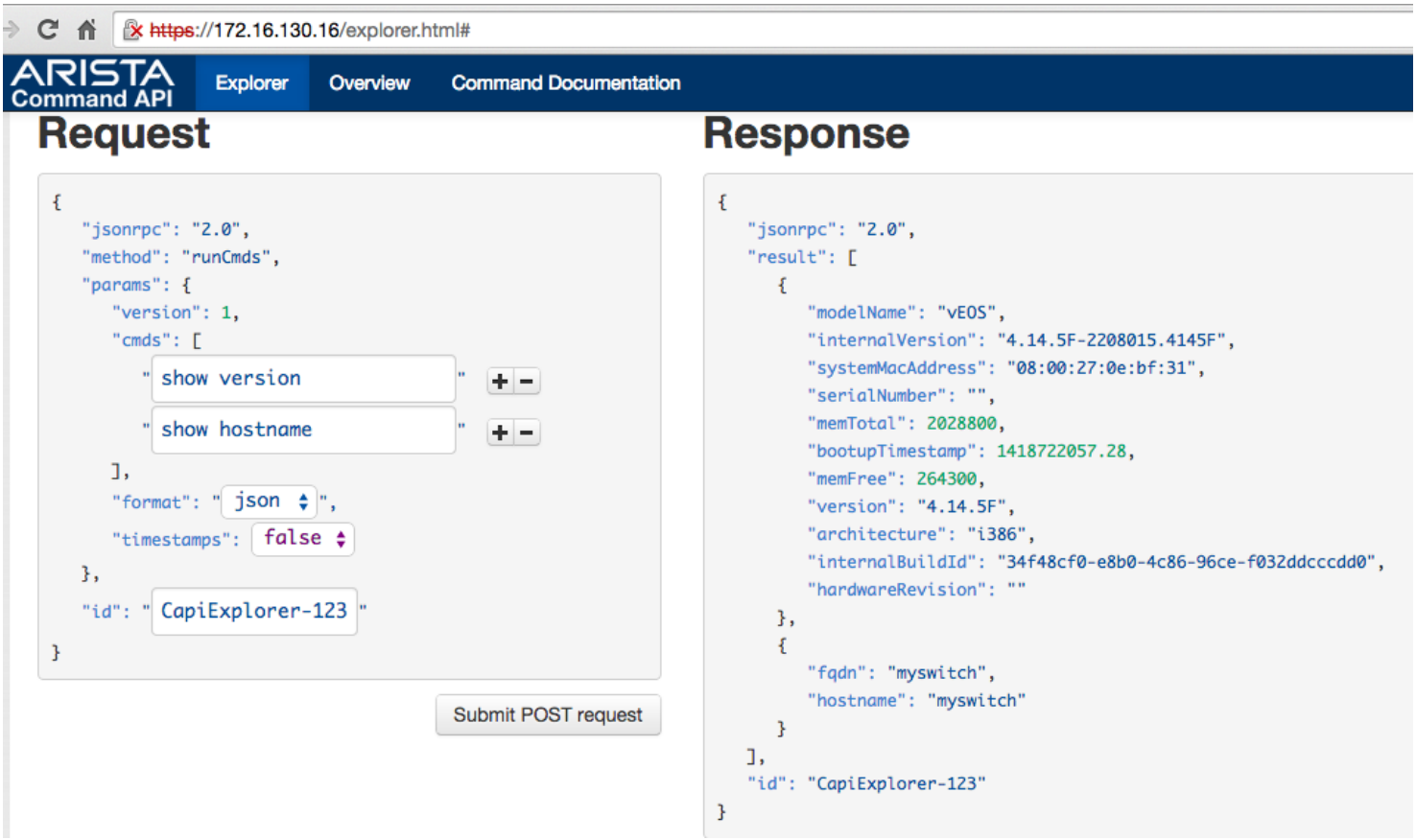
```
Arista> enable
Arista# configure terminal
Arista(config)# management api http-commands
Arista(config-mgmt-api-http-cmds)# [no] shutdown
Arista(config-mgmt-api-http-cmds)# [no] protocol https [port ]
Arista(config-mgmt-api-http-cmds)# [no] protocol http [port ]
```

In any CLI mode, you can issue the “show management api http-commands” command to view the status of the agent, including server information, attached users, last request time and other details.

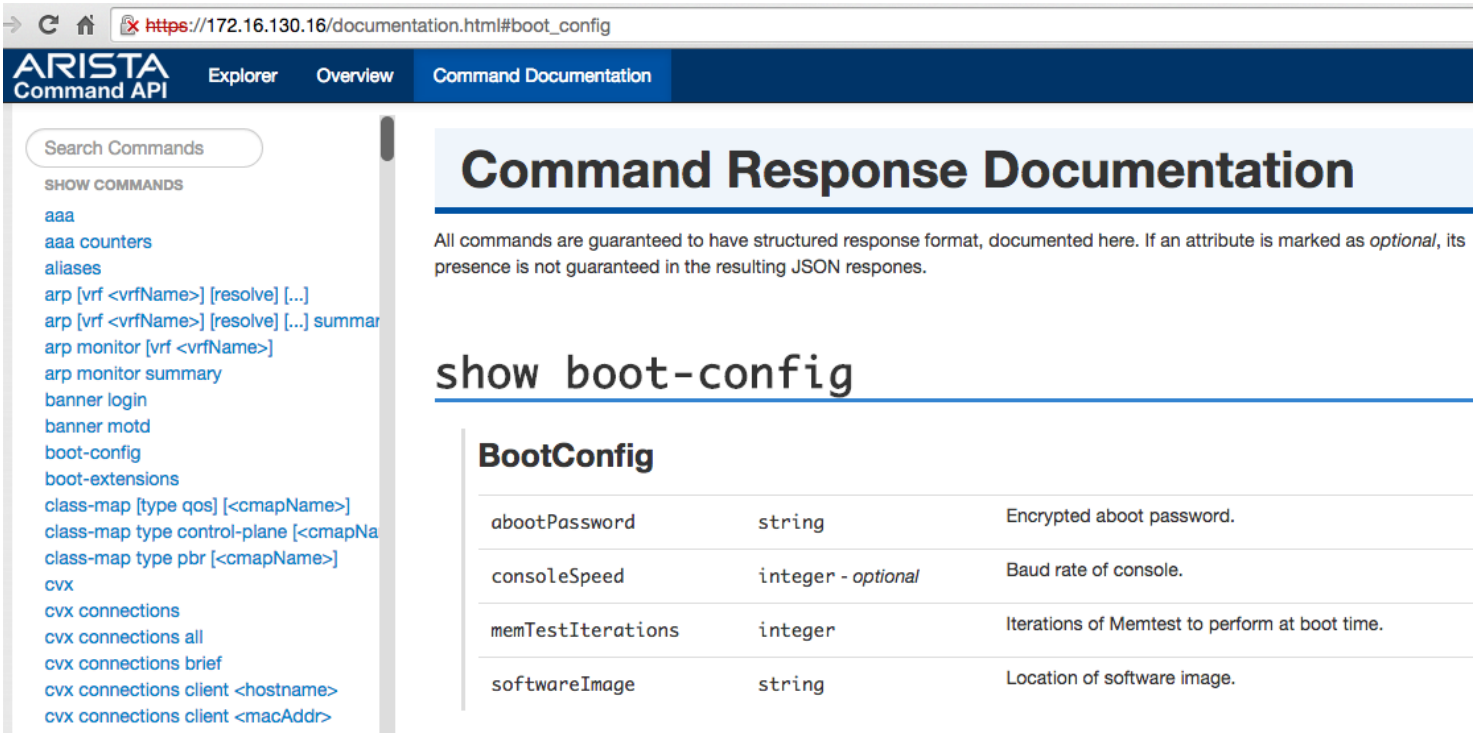
```
Arista# show management api http-commands
Enabled: Yes
HTTPS server: running, set to use port 443
HTTP server: shutdown, set to use port 80
Local HTTP server: shutdown, no authentication, set to use port 8080
Unix Socket server: shutdown, no authentication
VRF: default
Hits: 71
Last hit: 1433 seconds ago
Bytes in: 7669
Bytes out: 13554
Requests: 5
Commands: 8
Duration: 0.384 seconds
User        Requests       Bytes in       Bytes out       Last hit
----------- -------------- -------------- --------------- ----------------
admin       5              7669           13554           1433 seconds ago

URLs
----------------------------------------
Management1 : https://172.16.130.16:443
```

Step 2: Test the eAPI via web browsers: If the switch hasn't set user/pass, please configure the user/pass for switch remote access. Open your browser to access the **https://:port** as shown from the "show management api http-commands" output. Let's try common commands like "show version" and "show hostname":



We can see the output of these two commands formatted in JSON as shown. In the top right corner, we click the "Command Documentation" tab to get full list of CLIs supported and the corresponding output data entries definition.



Step 3: Use python scripts to test the eAPI: The Command API uses the lightweight, standardized JSON-RPC protocol to communicate between your program (the client) and the switch (the server), see http://www.jsonrpc.org/specification. Most languages have client libraries that abstract away the JSON-RPC internals, and there are some examples as below:

- Python JSON-RPC
- Perl JSON-RPC
- Javascript JSON-RPC
- And more JSON-RPC in Wiki

The quickest way is to start from your EOS, which has installed python and jsonrpclib already. Here is one example in python:

```
[admin@Arista flash]$ more hello.py
#!/usr/bin/python from jsonrpclib import Server
switch = Server( "https://admin:admin@172.16.130.16/command-api" )
response = switch.runCmds( 1, [ "show hostname" ] )
print "Hello, my name is: ", response[0][ "hostname" ]
response = switch.runCmds( 1, [ "show version" ] )
print "My MAC address is: ", response[0][ "systemMacAddress" ]
print "My version is: ", response[0][ "version" ]
```

In the script, this line defines the target. It is broken down as a URL with the following format:

<protocol>://<username>:<password>@<hostname or ip-address>/command-api

Alter this line as necessary to connect to the target switch. “/command-api” must always be present when using eAPI. The output looks as below:

```
[admin@Arista flash]$ ./hello.py
Hello, my name is: Arista
My MAC address is: 08:00:27:0e:bf:31
My version is: 4.14.5F
```

Step 4: Let’s take a look at some useful scripts example: Here is one example to apply the ACL to multiple switches, and the source code is put in the Github site. The script first uses the argparse library to get the parameters input, and then use linux editor for users to input ACL entries.

```
[admin@Arista flash]$ ./acledit.py -h
usage: acledit.py [-h] [--username USERNAME] [--password PASSWORD] [--https] ACL
SWITCH [SWITCH ...]
Edit Arista ACLs using your local editor

positional arguments:
 ACL Name of the access list to modify
 SWITCH Hostname or IP of the switch to query

optional arguments:
 -h, --help show this help message and exit
 --username USERNAME Name of the user to connect as
 --password PASSWORD The user's password
 --https Use HTTPS instead of HTTP
```

```
[admin@Arista flash]$ ./acledit.py --username admin --password admin --https acl101
172.16.130.16
No existing access list named acl101 - creating new ACL
```

Linux editor will be prompted up for input here:

```
permit ip any 172.16.0.0/16
permit ip 172.16.0.0/16 any
"/tmp/AclEditor-acl101" 3L, 57C written

New access list:
permit ip any 172.16.0.0/16
permit ip 172.16.0.0/16 any
Updating access list on switch 172.16.130.16 .... [SUCCESS]
Done!
```

Now we can check whether the ACL is created as expected.

```
Arista(config)# show ip access-lists
IP Access List acl101
10 permit ip any 172.16.0.0/16
20 permit ip 172.16.0.0/16 any
```

There are more scripts examples posted in the Github Arista-eosext section.

For some users familiar with Perl, here is a script example in Perl by our colleague John French from Arista POC team:

```
#!/usr/bin/perl
use JSON::RPC::Client;
use Data::Dumper;

sub cmd {
local ($url,*data,@cmds)=@_;
local $client = new JSON::RPC::Client;
local $host;

if ($url =~ m/http[^:]*:\/\/([^\/]+)\//) { $host=$1; }
else { return("couldn't extract hostname from url"); }

$client->ua->ssl_opts(
verify_hostname => 0
);
$client->ua->credentials(
$host, 'COMMAND_API_AUTH', 'admin' => ''
);
local $callobj = {
jsonrpc => "2.0",
method => 'runCmds',
params => {
version=>1,
cmds=>@cmds,
```

```
},
id=>"jfeapi"
};
$data = $client->call($url , $callobj);
if ($data) {
if ($data->is_error) {
return($data->error_message->{code}.": ".$data->error_message->{message});
}
else { return(0); }
}
else { return("connection error: ".$client->status_line); }
return("unexpected condition");
}

sub main {
# !!! PUT YOUR SWITCH URL HERE !!!
local $url = "https://sn218:443/command-api";
local $data;
local $err;

$err = &cmd($url,\$data,["enable","show version"]);
if (! $err) {
print $data->result->[1]{systemMacAddress},"\n";
}
else {
print "error: $err\n";
}
}

&main;
exit(0);
```